

AD-A261 320



WL-TR-93-1002

ADA COMPILER EVALUATION CAPABILITY
(ACEC) FINAL TECHNICAL REPORT RELEASE 3.0



THOMAS C. LEAVITT KERMIT TERRELL
BARBARA DECKER-LINDSEY SAM ASHBY
JULIE LEASTMAN

BOEING DEFENSE AND SPACE GROUP
P.O. BOX 7730, MS K80-13
WICHITA KS 67277-7730

JUN 1992

FINAL REPORT FOR 10/09/90-07/27/92

DTIC
ELECTE
FEB 26 1993
S C D

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

93-04018



AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT PATTERSON AFB OH 45433 -7409

93 2 25 048

NOTICE

WHEN GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA ARE USED FOR ANY PURPOSE OTHER THAN IN CONNECTION WITH A DEFINITELY GOVERNMENT-RELATED PROCUREMENT, THE UNITED STATES GOVERNMENT INCURS NO RESPONSIBILITY OR ANY OBLIGATION WHATSOEVER. THE FACT THAT THE GOVERNMENT MAY HAVE FORMULATED OR IN ANY WAY SUPPLIED THE SAID DRAWINGS, SPECIFICATIONS, OR OTHER DATA, IS NOT TO BE REGARDED BY IMPLICATION, OR OTHERWISE IN ANY MANNER CONSTRUED, AS LICENSING THE HOLDER, OR ANY OTHER PERSON OR CORPORATION; OR AS CONVEYING ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY IN ANY WAY BE RELATED THERETO.

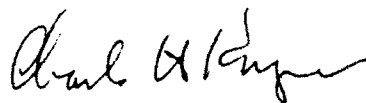
THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION



RAYMOND SZYMANSKI
Program Manager



TIMOTHY G. KEARNS, Maj, USAF
Chief
Readiness Technology Group



CHARLES H. KRUEGER, Chief
System Avionics Division
Avionics Directorate

IF YOUR ADDRESS HAS CHANGED, IF YOU WISH TO BE REMOVED FROM OUR MAILING LIST, OR IF THE ADDRESSEE IS NO LONGER EMPLOYED BY YOUR ORGANIZATION PLEASE NOTIFY WL/AAAF, WRIGHT-PATTERSON AFB, OH 45433-7409 TO HELP MAINTAIN A CURRENT MAILING LIST.

COPIES OF THIS REPORT SHOULD NOT BE RETURNED UNLESS RETURN IS REQUIRED BY SECURITY CONSIDERATIONS, CONTRACTUAL OBLIGATIONS, OR NOTICE ON A SPECIFIC DOCUMENT.

REPORT DOCUMENTATION PAGE

Form Approved

OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE JUN 1992	3. REPORT TYPE AND DATES COVERED FINAL 10/09/90--07/27/92	
4. TITLE AND SUBTITLE (ACEC) FINAL TECHNICAL REPORT RELEASE 3.0			FUNDING NUMBERS C F33615-87-C-1449 PE 63226 PR 2853 TA 01 WU 02	
5. AUTHOR THOMAS C. LEAVITT BARBARA DECKER-LINDSEY JULIE LEASTMAN			6. PERFORMING ORGANIZATION REPORT NUMBER	
7. AUTHOR (S) NAME(S) AND ADDRESS(ES) BOEING DEFENSE AND SPACE GROUP P.O. BOX 7730, MS K80-13 WICHITA, KS 67277-7730			8. SPONSORING MONITORING WLT-TR-93-1002	
9. AVIONICS DIRECTORATE (S) AND ADDRESS(ES) WRIGHT LABORATORY AIR FORCE SYSTEMS COMMAND WL/AAAF, Attn: SZYMANSKI 513-2556548			10. SPONSORING MONITORING WLT-TR-93-1002	
11. ACADA JOINT PROGRAM OFFICE (SPONSOR) 1211 S. FERN ST ARLINGTON VA 22202			THE ACEC HAS BEEN TRANSITIONED TO ASC/ SCSL, WPAFB, FOR LONG-TERM MAINTENANCE AND ENHANCEMENT. ACEC VERSION 4.0 EXPECTED COMPLETION IS IN APRIL 1993.	
12a. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.			12b. DISTRIBUTION CODE	

13. ABSTRACT (Maximum 200 words)

THIS REPORT DISCUSSES ACEC VERSION 3.0 AND COMPARES IT TO THE PREVIOUS RELEASES. IT ALSO DISCUSSES LESSONS LEARNED DURING THE DEVELOPMENT OF THE ACEC AND REVIEWS A NUMBER OF SIGNIFICANT FINDINGS.

14. ACADA, COMPILER, EVALUATION, BENCHMARKING. MIL-STD-1815A		15. NUMBER OF PAGES 29
16. SECURITY CLASSIFICATION UNCLASSIFIED		17. PRICE CODE UL

Table of Contents

1.	SCOPE	5
1.1	IDENTIFICATION	5
1.2	PURPOSE	6
1.3	INTRODUCTION	6
2.	APPLICABLE DOCUMENTS	8
2.1	GOVERNMENT DOCUMENTS	8
2.2	NON-GOVERNMENT DOCUMENTS	8
3.	ANALYSIS OF THE TEST SUITE	9
3.1	CONFIRMED EXPECTATIONS	17
3.1.1	Comparative Analysis (CA)	17
3.1.2	Single System Analysis	18
3.1.3	Include	19
3.1.4	Timing	19
3.1.5	Menu Interface For Analysis Tools	20
3.1.6	Guides	21
3.1.7	Assessors	22
3.1.8	Renaming	22
3.1.9	Pretest	22
3.2	PROBLEMS ENCOUNTERED	23
3.3	POSSIBLE ENHANCEMENTS	24
4.	SUMMARY	28
5.	NOTES	29
5.1	ABBREVIATIONS AND ACRONYMS	29

DTIC QUALITY INSPECTED 3

Accession For	
NTIS	<input checked="" type="checkbox"/>
CRA&I	<input type="checkbox"/>
DTIC	<input type="checkbox"/>
TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

1. SCOPE

This section identifies the Ada Compiler Evaluation Capability (ACEC) Release 3.0 product, states its purpose, and summarizes the purpose and contents of this Final Technical Report.

1.1 IDENTIFICATION

This is the Final Technical Report for the ACEC Release 3.0. It was developed by Boeing Defense & Space Group, Product Support Division under contract to the Wright Laboratory.

The first release of the ACEC program developed a Software Product consisting of a suite of benchmark test programs, support tools, a Reader's Guide, a User's Guide, and a Version Description Document (VDD).

The second release of the ACEC corrected problems found in the first release; approximately 300 new performance tests, assessors for program library systems, symbolic debuggers, and system diagnostics, a new tool to simplify the preparation of input to MEDIAN, and a Single System Analysis (SSA) tool; and upgraded supporting documentation to permit a user to assess the performance of Ada compilation systems.

The third release of the ACEC significantly improves usability and coverage. It adds:

- A systematic Pretest phase which guides the user through the adaptation necessary to run the ACEC on a new compilation system and the recording of the choices made for later reference.
- The user documentation has been enhanced, and provides more information to simplify execution of the ACEC and interpretation of the results.
- The sample command files have been modified in order to simplify the effort required to adapt to other Ada compilation systems.
- New performance tests have been added, including test problems: to determine the order of processing of alternatives in a SELECT statement; to determine the layout of arrays (row or column major); to report on the variability of the size of task control blocks, activation records, variant records and objects of an unconstrained type; to calculate task-switching time; to report whether a compiler will evaluate an arithmetic expression in a way that will produce a result different from the canonical order; to provide an example application of an inference engine; to systematically evaluate the variation in performance as the size of a section of code increases; to compare differences in coding style based on IF statements versus exceptions; and to explore the run-time overhead of entering blocks that declare uninitialized variables.
- The ACEC performance tests have been renamed, repackaged and reorganized into groups and subgroups by test objective. This is to give the users an easier comprehension of the over 1600 test elements and to provide test result analysis flexibility. This reorganization, along

with added functionality, allows the user to assign weights for individual tests and for groups of related tests for analysis purposes.

- The data-gathering and analysis tasks have been automated, eliminating many of the time-consuming and cumbersome compile, link and run steps in previous versions. The Comparative Analysis tool provides confidence intervals on system factors between problems within groups of related tests, and between the factors calculated for different groups.
- Capacity tests for both run time and compile/link time have been added.
- Systematic Compile Speed tests have been added. These tests are designed to test the sensitivity of compile speeds to different language feature usage.

1.2 PURPOSE

The purpose of this document is to review the problems encountered and the lessons learned in the process of developing the third release of the ACEC Software Product.

The descriptions on how to use the product have been presented in the Guides. Information contained in the Guides will not be repeated here.

The numeric results of the ACEC Release 3.0 testing are presented in the ACEC Software Test Report for Release 3.0 and will not be repeated here.

1.3 INTRODUCTION

The ACEC group in Boeing Defense & Space Group, Product Support Division, Avionics Systems and Software Department, tested the ACEC Release 3.0 in November - December 1991. The ACEC was tested on five trial systems, 3 of which were the same as were used to test ACEC Release 2.0: DEC Ada VAX/VMS self-hosted; TeleSoft VAX/VMS self-hosted; and the Verdix self-hosted system for the Silicon Graphics (a UNIX-based system). The VAX/VMS hosted 1750A targeted compilation system used in Release 2.0 was replaced with the TLD VAX/VMS hosted 1750A targeted compilation system (using a different target processor). The Meridian self-hosted compilation system for the DECStation (another UNIX-based operating system) was used in place of the ALSYS self-hosted compilation system on the Apollo.

The ACEC Release 3.0 Software Product consists of:

- A set of tests and procedures forming a Pretest which guides an ACEC user step-by-step through the process of adapting the ACEC product to a new compilation system.
- A suite of performance test programs.
- A set of supporting packages.

These include a preprocessor to incorporate the timing loop in the test programs, a portable implementation of a math library, and packages defining global types, variables, and subprograms for use in the test suite and analysis tools.

- A set of analysis tools.

These include CONDENSE, MENU, Comparative Analysis (CA), and Single System Analysis (SSA).

- A set of assessors for diagnostics, symbolic debuggers, program library systems, and compilation system capacity limits.
- Sample command files to compile and execute the tools and tests for a VMS and for a UNIX-based system which can be used as guides for porting to other systems.

This report assumes that the reader is familiar with the ACEC Release 3.0 User's Guide, the Reader's Guide, and the Version Description Document (VDD). The reader should refer to the VDD Release 3.0 for a description of the tests. Information from the Software Test Report Release 3.0 is also referenced.

This document reviews the development of the third release of the ACEC Software Product. It gives an analysis of results and a review of the lessons learned.

2. APPLICABLE DOCUMENTS

The following documents are referenced in this guide.

2.1 GOVERNMENT DOCUMENTS

MIL-STD-1815A Reference Manual for the Ada Programming Language (LRM)

2.2 NON-GOVERNMENT DOCUMENTS

D500-12563-1 Ada Compiler Evaluation Capability (ACEC)
Version Description Document (VDD) Release 3.0
Boeing Defense & Space Group
Product Support Division
P.O. Box 7730
Wichita, Kansas

D500-12564-1 Ada Compiler Evaluation Capability (ACEC)
Technical Operating Report (TOR)
User's Guide Release 3.0
Boeing Defense & Space Group
Product Support Division

D500-12565-1 Ada Compiler Evaluation Capability (ACEC)
Technical Operating Report (TOR)
Reader's Guide Release 3.0
Boeing Defense & Space Group
Product Support Division

D500-12570-1 Ada Compiler Evaluation Capability (ACEC)
Software Test Report Release 3.0
Boeing Defense & Space Group
Product Support Division

3. ANALYSIS OF THE TEST SUITE

The following table displays the test problems where one of the trial systems used in FQT had a residual for execution time of less than or equal to a tenth, or greater than or equal to ten. Such highly variable results highlight test problems where systems may have taken very different approaches, and may identify problems where systems are not performing comparable operations. The trial systems are named V_91, T_91, S_91, M_91, and D_91, as discussed in the ACEC Software Test Report Release 3.0. The full set of numeric results obtained by CA on the trial system data are presented in the Software Test Report Release 3.0 and will not be duplicated here.

Some test problems are mentioned more than once because they had exceptional residuals on more than one compilation system

PROBLEM NAME

RESIDUAL	
io_sq_io_copy_02	0.01
dr_ba_bool_arrays_29	0.01
io_sq_io_copy_01	0.01
dr_aa_array_aggreg_01	0.01
dr_ao_array_oper_20	0.01
io_sq_io_scan_11	0.01
dr_aa_array_aggreg_01	0.01
dr_ba_bool_arrays_28	0.01
io_sq_io_copy_02	0.02
dr_ba_bool_arrays_13	0.02
io_sq_io_copy_01	0.02
dr_ba_bool_arrays_14	0.02
dr_ba_bool_arrays_15	0.02
dr_ba_bool_arrays_17	0.02
dr_ba_bool_arrays_32	0.02
io_sq_io_scan_12	0.02
io_ds_io_04	0.03
dr_be_bool_express_12	0.03
dr_be_bool_express_12a	0.03
dr_aa_array_aggreg_01	0.03
io_tf_text_ioflt_str_03	0.03
dr_ba_bool_arrays_29	0.03
dr_ba_bool_arrays_11	0.03
dr_aa_array_aggreg_01	0.03
op_fa_alias_08	0.03
dr_ba_bool_arrays_17	0.04
dr_ba_bool_arrays_12	0.04
io_da_io_scan_08	0.04
op_fa_alias_12	0.04

io_sq_io_17	0.04
io_tf_text_ioflt_str_01	0.04
op_lv_loop_invar_08	0.05
io_tf_text_ioflt_str_02	0.05
do_rp_rep_pack_24	0.05
dr_ba_bool_arrays_15	0.05
op_fs_fold_simp_24	0.05
dr_ba_bool_arrays_28	0.05
po_pa_d_library_07	0.05
io_dp_io_pattern_07	0.05
do_rp_rep_pack_49	0.05
do_rp_rep_pack_19	0.05
do_rp_rep_pack_54	0.05
op_fs_fold_simp_17	0.06
io_du_io_unif_05	0.06
dr_ba_bool_arrays_32	0.06
op_oe_order_of_eval_20	0.06
io_da_io_scan_15	0.06
dr_ba_bool_arrays_29	0.06
io_dp_io_pattern_08	0.06
dr_ba_bool_arrays_14	0.06
dr_te_type_enum_08	0.06
dr_ba_bool_arrays_23	0.06
op_fs_fold_simp_21	0.06
op_lv_loop_invar_09	0.06
io_ti_text_io_int_str_02	0.06
io_da_io_scan_04	0.06
io_tf_text_ioflt_str_01	0.06
io_tf_text_ioflt_str_03	0.07
io_dp_io_pattern_06	0.07
ms_ec_express_cat_01	0.07
io_dr_io_recur_01	0.07
io_di_io_80_20_03	0.07
io_dp_io_pattern_07	0.08
io_tx_io_01	0.08
dr_ba_bool_arrays_13	0.08
dr_ba_bool_arrays_14	0.08
xh_er_except_raise_03	0.08
io_di_io_80_20_05	0.08
op_lv_loop_invar_01	0.08
io_dr_io_recur_01	0.08
io_dp_io_pattern_08	0.09
io_tx_io_09	0.09
op_as_alge_simp_06	0.09
io_tx_io_24	0.09
dr_ba_bool_arrays_15	0.09

do_cu_conv_unck_01	0.09
ms_ec_express_cat_01	0.09
dr_ao_array_oper_33	0.10
dr_be_bool_express_12a	0.10
st_is_if_code_style_29	0.10
dr_ao_array_oper_32	0.10
dr_ba_bool_arrays_05	0.10
dr_ba_bool_arrays_07	0.10
dr_ba_bool_arrays_11	0.10
dr_rd_rec_discr_02	0.10
st_is_if_code_style_29	11.34
io_tx_io_09	12.17
io_tx_inst_05	12.41
io_tx_inst_03	13.54
io_tx_inst_02	19.16
io_tx_inst_01	19.20
io_tx_inst_04	24.12

The following alphabetic list of performance problem names describes the features in each of these problems and the systems which were flagged as exceptional.

PROBLEM NAME	APPARENT REASON
do_cu_conv_unck_01	This problem tests unchecked conversions by passing a (converted) array parameter to a function. No explicit code is required to support the conversion. V_91 was exceptionally slow and S_91 was exceptionally fast. The low subprogram overhead on S_91 contributed to its high performance on this problem.
do_rp_rep_pack_19	This is a test of fetching elements from a packed array. M_91 was exceptionally fast on this problem, perhaps because the system did NOT pack the array, permitting faster access than the other systems.
do_rp_rep_pack_24	This is a test of fetching elements from a packed array. M_91 was exceptionally fast on this problem, perhaps because the system did NOT pack the array, permitting faster access than the other systems.
do_rp_rep_pack_49	This is a test of referencing elements from packed and unpacked arrays. M_91 was exceptionally fast on this problem, perhaps because the system did NOT pack the array, permitting faster

do_rp_rep_pack_54	access than the other systems. This is a test of fetching elements from a packed array. M_91 was exceptionally fast on this problem, perhaps because the system did NOT pack the array, permitting faster access than the other systems.
dr_aa_array_aggreg_01	This problem declares a constant array initialized with literals. M_91 does it particularly poorly and V_91, T_91, S_91 and D_91 (by comparison to the mean time for the problem) do it particularly well.
dr_ao_array_oper_20	This problem deals with initializing a constant array with another array (which might be optimized by effectively renaming). D_91 was exceptionally fast and M_91 was exceptionally slow on this problem.
dr_ao_array_oper_32	This problem makes repeated accesses to an element of a 2-D array whose actual subscripts are nested FOR loop indexes. This is an important special case of strength reduction. M_91 did exceptionally well.
dr_ao_array_oper_33	This problem makes repeated accesses to an element of a 2-D array whose actual subscripts are nested FOR loop indexes. This is an important special case of strength reduction. M_91 did exceptionally well.
dr_ba_bool_arrays_05	This problem tests boolean OR operation on a small unpacked boolean array. V_91 was exceptionally fast. Apparently both it and T_91 did NOT call on run-time library routines to perform the work.
dr_ba_bool_arrays_07	This problem tests boolean XOR operation on a small unpacked boolean array. V_91 was exceptionally fast. Apparently both it and T_91 did NOT call on run-time library routines to perform the work.
dr_ba_bool_arrays_11	This problem performs boolean operations on small packed boolean arrays. T_91 and S_91 were both exceptionally fast (only a small number of instructions were performed).
dr_ba_bool_arrays_12	This problem performs boolean operations on small packed boolean arrays. T_91 was exceptionally fast

dr_ba_bool_arrays_13	<p>requiring only a small number of instructions to be executed.</p> <p>This problem performs boolean operations on small packed boolean arrays. T_91 and S_91 were exceptionally fast requiring only a small number of instructions to be executed. M_91 was exceptionally slow.</p>
dr_ba_bool_arrays_14	<p>This problem performs a simple boolean AND on two small packed boolean arrays. V_91, T_91 and S_91 were exceptionally fast requiring only a small number of instructions to be executed.</p>
dr_ba_bool_arrays_15	<p>This problem performs a simple boolean OR on two small packed boolean arrays. V_91, T_91 and S_91 were exceptionally fast requiring only a small number of instructions to be executed.</p>
dr_ba_bool_arrays_17	<p>This problem performs a simple boolean XOR on two small packed boolean arrays. V_91 and T_91 were exceptionally fast requiring only a small number of instructions to be executed.</p>
dr_ba_bool_arrays_23	<p>This problem performs simple boolean AND and "=" operations on a long unpacked boolean array. T_91 supports it exceptionally well, perhaps with inline code.</p>
dr_ba_bool_arrays_28	<p>This problem performs an OR operation on a packed boolean array of size 16. V_91 and T_91 were exceptionally fast (and S_91 was also fairly fast), each taking a few instructions to perform the test problem.</p>
dr_ba_bool_arrays_29	<p>This problem is similar to dr_ba_bool_arrays_28 but uses for one operand a named aggregate with literal values rather than a simple variable. For all but M_91, the times of the problems are indistinguishable (as was hoped). V_91, T_91 and S_91 were exceptionally fast, and M_91 was exceptionally slow (apparently constructing the aggregate at run time).</p>
dr_ba_bool_arrays_32	<p>This problem is similar to dr_ba_bool_arrays_28 but uses for one operand a subscripted array reference,</p>

	rather than a simple variable.
	V_91 and T_91 were exceptionally fast, and S_91 was only slightly slower.
dr_be_bool_express_12	This problem performs bit manipulation using array indexes on a packed array. T_91 is exceptionally fast.
dr_be_bool_express_12a	This problem performs bit manipulation using array-wide logical operators on a packed array. T_91 and S_91 are exceptionally fast.
dr_rd_rec_discr_02	This problem checks a discriminated record and raises CONSTRAINT_ERROR. D_91 was exceptionally fast at processing the exception. T_91 was almost exceptional.
dr_te_type_enum_08	This test problem explores using representation clauses to specify values for an enumerated type. D_91 was exceptionally fast.
io_da_io_scan_04	This is an I/O operation where large differences between systems are expected and where caching and buffering and general Operating System I/O processing greatly influences performance. M_91 was exceptionally fast.
io_da_io_scan_08	Direct file I/O operation test. M_91 was exceptionally fast.
io_da_io_scan_15	Direct file I/O operation test. S_91 was exceptionally fast.
io_di_io_80_20_03	Direct file I/O operation test on a file with 10_000 records. M_91 was exceptionally fast.
io_di_io_80_20_05	Direct file I/O operation test on a file with 100_000 records. M_91 was exceptionally fast.
io_dp_io_pattern_06	Direct file I/O operation test. S_91 and M_91 were exceptionally fast.
io_dp_io_pattern_07	Direct file I/O operation test. S_91 and M_91 were exceptionally fast.
io_dp_io_pattern_08	Direct file I/O operation test. S_91 and M_91 were exceptionally fast.
io_dr_io_recur_01	Direct file I/O operation test. S_91 and M_91 were exceptionally fast.
io_ds_io_04	I/O operation. M_91 was exceptionally fast.
io_du_io_unif_05	Direct file I/O operation test reading from file with 100_000 records. S_91

io_sq_io_17	was exceptionally fast.
io_sq_io_copy_01	Open and close a sequential file. M_91 was exceptionally fast.
io_sq_io_copy_02	Sequential file I/O operation test. V_91 and M_91 were exceptionally fast.
io_sq_io_scan_11	Sequential file I/O operation test. V_91 and M_91 were exceptionally fast.
io_sq_io_scan_12	Sequential I/O operation test, writing variable length records. V_91 was exceptionally fast.
io_tf_text_ioflt_str_01	Sequential I/O operation test, reading variable length records. V_91 was exceptionally fast.
io_tf_text_ioflt_str_02	Put to a string. Test of FLOAT_IO conversions. V_91 and M_91 were exceptionally fast.
io_tf_text_ioflt_str_03	Put to a string. Test of FLOAT_IO conversions. V_91 was exceptionally fast.
io_ti_text_io_int_str_02	Put to a string. Test of FLOAT_IO conversions. V_91 and M_91 were exceptionally fast.
io_tx_inst_01	Put to a string. Test of INTEGER_IO conversions. V_91 was exceptionally fast.
io_tx_inst_02	Instantiate enumeration I/O. D_91 was exceptionally slow.
io_tx_inst_03	Instantiate enumeration I/O. D_91 was exceptionally slow.
io_tx_inst_04	String assignment. D_91 was exceptionally slow.
io_tx_inst_05	Use of 'image attribute. D_91 was exceptionally slow.
io_tx_io_01	Literal string assignment. D_91 was exceptionally slow.
io_tx_io_09	Open and close a text file. M_91 was exceptionally fast.
io_tx_io_24	I/O operation, reset file. S_91 was exceptionally slow. M_91 was exceptionally fast.
ms_ec_express_cat_01	Console I/O operation. V_91 was exceptionally fast.
op_as_alge_simp_06	This problem uses the string concatenation operations and slices. V_91 and T_91 were exceptionally fast.
	The problem contains a multiple by a literal "1." It was exceptionally fast on V_91, because it folded the operation.

op_fa_alias_08	This problem contains complex foldable code. T_91 was exceptionally fast.
op_fa_alias_12	This problem contains complex foldable code. T_91 was exceptionally fast.
op_fs_fold_simp_17	This problem contains code with foldable subexpressions. S_91 was exceptionally fast, and T_91 was also fairly fast.
op_fs_fold_simp_21	This problem contains code with foldable subexpressions, if a compiler propagates a literal assignment to a variable declared with a CONSTANT attribute. S_91 was exceptionally fast.
op_fs_fold_simp_24	This problem contains foldable literal subexpressions. S_91 was exceptionally fast.
op_lv_loop_invar_01	This problem contains a FOR loop with one invariant assignment within it. It could be optimized into one assignment, removing the invariant code from the loop and removing the (now) redundant loop overhead. M_91 was exceptionally fast.
op_lv_loop_invar_08	This problem tests several cases where applying loop invariant motion could be applied to reordering expressions across parentheses. V_91 was exceptionally fast.
op_lv_loop_invar_09	This is a version of op_lv_loop_invar_08 using type FLOAT9.
op_oe_order_of_eval_20	One of the systems uses extended precision temporaries to evaluate an expression and in so doing avoids raising an exception (exception raising can be expensive). This is a valid comparison test, even though the problem will raise an exception on some systems and not on others because the intent of the test is to determine precisely this behavior. S_91 was exceptionally fast.
po_pa_d_library_07	This problem tests for elaborating a package declaration which requires dynamically-sized storage. M_91 was exceptionally fast.
st_is_if_code_style_29	This problem raises a CONSTRAINT_ERROR exception when converting to a user-defined subtype. T_91 was exceptionally fast.
xh_er_except_raise_03	This problem declares a block with an exception handler for a user-

defined exception. The exception is NOT raised, so the problem is a measure of block entry and exit time. S_91 was exceptionally fast (and V_91 was almost exceptional.

In this list of exceptional test problems, I/O operations occur frequently, followed by operations on packed boolean arrays and (to a lesser extent) exception processing. Examining the particular systems marked as exceptional (see Test Report for details) shows that it is not true that compilation systems follow a simple linear ranking, with all systems optimizing the "simple" problems and the better systems optimizing more of the "harder" problems. A system which does not generally optimize many problems is sometimes the only system to optimize one particular problem. This behavior may not match a priori expectations, but it does reflect actual behavior.

3.1 CONFIRMED EXPECTATIONS

The following subsections list design decisions which worked well.

3.1.1 Comparative Analysis (CA)

In analyzing results from several systems, some type of statistical analysis tool such as CA is a practical necessity for a test suite with more than a small number of test problems. The analysis focuses the attention of the ACEC users on the test problems with "unusual" results. It permits a *form of report-by-exception* where ACEC users can concentrate their attention on the test problems with anomalous performance. Since most test problems will not be flagged by CA as outliers, ACEC users will be able to "skim" over most of them and focus on those where large differences between systems were observed.

Without any comparative analysis tool, a test suite would require users to "understand" each of the test problems, at least to know if a result on one system was good, bad, or indifferent. The residual matrix is very helpful since it establishes a normalized metric for test results where a residual close to one is "typical." No analysis tool can extract more information from a set of data than is implicit in the collection of "raw" measurements, but it can make the relationships between data more apparent. It would be very easy to overlook the fact that a system executes some test problems twice as fast as typical when all the data is presented as one large table of timing measurements. It is important to prevent users from being overwhelmed by the volume of data and CA serves this role.

Changes added to CA in Release 3.0 provided features which should be useful:

- **Analysis by groups**

The CA tool now performs its analysis on groups of related problems rather than on the entire set of test problems, and does a summary analysis over the groups. This will permit users to more easily discern the relative performance of systems on the (preselected) groups. For example, comparing system performance on I/O-intensive test problems is more straightforward.

ward now than in Release 2.0 because Release 3.0 has identified a set of I/O test problems and CA will compare these sets of problems between systems.

- Confidence intervals

The CA tool now calculates confidence intervals rather than point estimates for system factors, and will identify whether differences in system factors found by CA are statistically significant. Using the Release 2.0 MEDIAN tool, some users may have concluded that one system was better than another because the system factor was better, when in fact the difference between the systems was not statistically significant. This was the major reason for the change in the statistical analysis algorithm.

- Initial report summary

Several changes were made to the CA reports to help users. Provisions were made for users to provide descriptive text about the systems being compared, in order to more clearly identify the system on the CA report. Users are encouraged to provide version numbers of compilers and operating systems and to describe the hardware configuration tested. System factors and confidence intervals are displayed graphically. Graphics also show whether there is a statistically significant difference between computed system factors.

CA shows the count of errors of each type on each system. Error data is important because a fast system which doesn't support the full Ada language is not necessarily "better" than a slower system which supports more of the language.

- Weights

CA now provides the capability for users to provide weights for individual groups, or for individual test problems.

- Performance

It is now considerably faster to re-analyze sets of data, including and excluding different systems, in CA than it was with MEDIAN in Release 2.0. In Release 3.0, it is not necessary to invoke the Ada compiler to process the performance data. This enhanced performance in reporting should encourage more exploration of the data that the users collect.

The analysis by groups and the confidence levels fill a need reported by a number of ACEC users.

3.1.2 Single System Analysis

The Single System Analysis (SSA) tool was not substantially changed from Release 2.0 (other than reflecting the renaming of test problems). It still provides an automated way of comparing and displaying results from sets of related performance test problems. ACEC users had to manually compare results of related tests before the development of this tool, however, the speed and automated nature of the tool make such comparisons much simpler.

3.1.3 Include

The Include tool (`zg_incl`) is an ACEC support tool used to textually expand Ada source text. It is used to insert the timing loop code into the test programs. Refer to the User's Guide for a discussion on the use of this tool.

It permits modification of the timing code:

- To accommodate measuring either Central Processing Unit (CPU) time or elapsed time.
- To accommodate the lack of support for the label `'ADDRESS` attribute.

The flexibility provided by the Include processing has proved valuable in Release 3.0 because significant changes were made to the timing loop (refer to the next section) by changing a few files. It was not necessary to modify the source code bracketing all 1627 test problems.

3.1.4 Timing

The timing loop code is responsible for measuring the execution time and code expansion for each test problem. ACEC users wishing to construct their own test problems will find the conventions for using the timing loop and how to isolate language features to be tested are discussed in the Reader's Guide.

The timing loop code was changed from the second release in several ways:

- **INITTIME**

The approach to initializing timing loop parameters was changed. In Release 2.0, the initialization code was inserted by Include into every test program and as a result was compiled and executed many times (once for each test program). In Release 3.0, the initialization code is compiled and executed once for each compilation option (optimization, checking, no-optimization, and checking suppressed). The initialization code in each test program copies the timing loop parameters from files created during Pretest (see section 3.1.9 for a discussion of the ACEC Pretest) and then inserts a section of code which will measure the time to execute a sample "null" loop and verify that the copied timing loop parameters are consistent with the measurements of this sample loop. If this verification checking were not performed, wrong timings could be obtained because different compilation options (suppression of checking, optimization levels, etc.) can change the values of the timing loop parameters.

- This approach reduces the total lines of code passed through the compiler (the verification code is much smaller than the initialization code was in Release 2.0). The amount of time spent executing initialization code in each test program is also reduced because one verification loop can execute much faster than the "full" initialization code.

An option is provided to use `zg_incl` to incorporate the "full" initialization code, as was done in Release 2.0, for users who wish to explore the performance effects of systems with many compiler options.

- **EXCESSIVE_TIME**

All the test programs in the Storage_Reclamation_Implicit subgroup in Release 3.0 make an initial measurement to estimate the time to run the problem to completion (the code is executed 100_000 times). If this estimate is larger than the value EXCESSIVE_TIME, the test problem execution is skipped and a special error code written.

In Release 2.0, only seven of the problems observed to exceed time limits on the trial systems included this check. The Release 3.0 approach will be safer in skipping long running test problems unless the user explicitly modifies code to permit longer running problems to complete.

- **EXCESSIVE_VARIABILITY**

In order to reduce the number of test problems which get unreliable measurement error codes, the value of EXCESSIVE_VARIABILITY was reduced from 2.0 to 1.25. This will have the effect of making it more likely that the timing loop control logic will detect excessive variability and "jump" to a larger value of the inner loop count, producing fewer unreliable measurements.

A reduction in the unreliable measurement error codes was observed on the trial systems.

- **ERR_LARGE_NEGATIVE_TIME**

The timing loop code uses a dual-loop approach which calculates the execution time for a test problem by subtracting the null loop time from the elapsed time per iterations. When the elapsed time per iteration is less than the null loop time, a spurious negative time measurement might be indicated. When this happened in release 2.0, a zero time was reported when the negative time was less than the variation in the null loop time and an unreliable time was reported otherwise. This was changed in release 3.0 so that a new error code (ERR_LARGE_NEGATIVE_TIME) is returned when the elapsed time per iteration is significantly less than the null loop time.

The changes made in Release 3.0 will both enhance the efficiency of gathering data (it should take less time to collect the measurements) and increase the chances that reliable data will be collected.

The compile and link times in Release 3.0 are measured and output by Ada programs rather than job control procedures. This permits easier adaptation and provides for automated collection of data for analysis because the format of the compilation/link time data is implementation-independent (it is output as a value of the predefined time DURATION).

3.1.5 Menu Interface For Analysis Tools

The new interactive MENU interface to the analysis tools provided in Release 3.0 provides benefits in several areas:

- It is easier to use.

The MENU interface to the analysis tools is easier to use and faster to learn than the manual manipulations which would have been required to perform the corresponding options in Release 2.0.

The MENU interface should increase the confidence of new ACEC users who will be able to obtain useful results with less effort than with prior releases.

- It is faster.

The CA tool now runs much faster because a new analysis will not involve a recompilation and link. The faster execution time of the tool will make interactive use feasible and encourage ACEC users to explore more analysis options.

- It eliminates some misleading results.

The CONDENSE tool (comparable to Release 2.0 FORMAT tool) reads logs and database files and produces report files showing missing data, erroneous problems and duplicate problems. It includes consistency checking to avoid some types of errors. For example, in release 2.0 the compilation time was always reported. A compiler might fail quickly and reject a valid ACEC program. This behavior distorted the analysis of compilation speeds because it made a system which failed quickly look fast when analyzing compilation times. In release 3.0, the compilation time for a program is set to an error code when there is no valid execution time for the test problems contained within it.

3.1.6 Guides

The writing of effective user documentation is difficult and time consuming. It is also very important to the usability of a product, particularly when there is no provision for telephone "hot-lines" for helping users who run into problems.

The ACEC user documentation is extensive.

The User's Guide presents a step-by-step guide to using the ACEC. It now includes a Troubleshooting Guide appendix discussing common problems, symptoms and workarounds. It includes copies of all the assessor summary report forms.

The Reader's Guide discusses issues involved in understanding the results produced by the test suite, discusses background issues in evaluation and benchmarking, and provides guidelines for users writing their own test problems.

The ACEC Version Description Document (VDD) contains useful information about the test suite itself, descriptions of the individual test problems and descriptions of the assessor scenarios.

The revised documents were well received by the beta test sites.

3.1.7 Assessors

Release 3.0 of the ACEC includes assessors for the Ada program library management system, the diagnostic messages, the symbolic debugger and system capacities.

The Capacity Assessor is new in Release 3.0 and contains 32 compile-time and 9 run-time feature tests. The Capacity Assessor requires some system-dependent adaptation in the host system command files which was not particularly difficult on any of the trial systems. When running the Capacity Assessor with limits large enough to find maximum system capacities, the tests can take a long time to complete; however, this phenomena is expected on systems which have large limits.

The assessors have generally worked well, however, they do require more programmer effort than we might have liked because the ACEC user must manually adapt them to each system being evaluated.

3.1.8 Renaming

All the test problems and tools have been renamed and repackaged to reflect logical groupings of related programs.

The performance tests have been organized into groups and subgroups (reflected in the first five characters of the file name) making it easy to select subsets of test problems to run.

In general, each performance test is now in a separate compilation unit. Performance test programs WITH a group of procedures containing individual tests and then call on them. Before compiling the test problem procedures, a file with the same name is compiled which will write an error code indicating compile-time failure if it is called. If there is an error in compiling the "real" problem, the "dummy" version which writes an error code will be linked into the test program. This technique has worked out well in permitting automated processing of many problems with compile-time errors. It also reduces the number of packaging errors.

3.1.9 Pretest

A Pretest section has been added which leads the user step-by-step through the process of adapting the ACEC to a new compilation system and verifying that needed facilities are present. It checks the accuracy of the system clock, the math library, determines timing loop parameters, checks if label ADDRESS is supported, and compiles the baseline files, the programs which write compiler/linker time stamps, and the analysis tools.

Beta testers reported they particularly liked this addition. It can reduce anxiety of users adapting the ACEC by helping them avoid blind alleys in adapting the ACEC to a new compilation system. The summary report form provides a document for recording the changes made in adapting the ACEC and remembering the configuration tested. One user of an earlier version of the ACEC reported that he didn't find out that the analysis tools would not compile and run on the system he was evaluating until he had run all the performance tests and did not have enough time left on his evaluation schedule to develop an alternative. With release 3.0 he would have

known about the problem earlier and might have developed alternatives such as getting a corrected version from the implementor of the compiler, modified code in the analysis tools to work around problems in the compiler, or arranged to use a different compilation system to perform the analysis.

3.2 PROBLEMS ENCOUNTERED

During the development of the ACEC, the test suite and tools were executed on multiple systems both to verify portability of the code, and to provide sample data to demonstrate the Comparative Analysis tools. These are the "trial" systems referred to throughout this report. During this process, some implementation errors and restrictions were discovered in the trial systems, as detailed below:

- Some systems did not support the label ADDRESS clause (used in the ACEC to measure code expansion size).
- Many systems do not support tying tasks to interrupts.
- Several systems do not support a type SMALL specifying a fixed point delta which is NOT a power of two.
- Many systems did not support asynchronous I/O operations - that is, an I/O operation in any task causes the program to halt until the I/O completes.
- Some systems failed to reclaim implicitly allocated space. The LRM does NOT require that the space be reused, so this is an implementation restriction rather than an error, but may be a serious problem to any program which needs to run for a long time.
- Some systems restrict the type of files that can be processed. Test problems using a SEQUENTIAL_IO instantiated with an unconstrained type (that is, STRING) were not accepted by many systems, as were problems using sequential and direct files with a variant record. This feature is not supported by many of the trial systems. On the DEC Ada system the test problems would not run using the default FORM strings and a system-dependent adaptation was necessary.
- The test problem which calls on an assembly language procedure did not work on all systems. In several cases, the trial systems claim to support the facility, but the provided documentation was unclear, and the initial attempt to adapt the test problem failed.
- Capacity limitations - some programs were too large to be compiled by some of the systems.
- The implementation-dependent type SYSTEM.ADDRESS is converted to an integer type to calculate sizes. The size of this integer type is implementation dependent and may need to be adapted for use on different systems. The need for adaptation is awkward.

- Some implementations imposed restrictions on length clauses. Several only supported the declaration of specific predefined sizes. For example, several would not accept a specification for a three-bit-wide field.
- Pre-emptive priority scheduling was not supported on all the trial systems. This caused some of the test problems to report a run-time error.
- One system (for an embedded target) did not support any file I/O.
- Some systems did not support an option to specify tasking discipline (time-sliced versus run-till-blocked). This is NOT required by the LRM, but some classes of programs find it helpful.
- On some systems the algorithm used to choose among the open alternatives in a SELECT statement is not "fair." (Some open alternatives do not get chosen at all on these systems.) Fairness is NOT required by the LRM, but some classes of programs find it helpful.
- A closure (recompilation) facility is required for some of the problems in the Systematic Compile Speed Group (SY). This was not available on all trial systems.
- Verification errors were fairly common on the Silicon Graphics system.

The problem with verification errors on the Silicon Graphics seems to be largely due to a fault in the command files. When the command file was corrected, results changed from 438 verification failures and 135 large negative times to 43 verification failures and no large negative times.

Most of these problems are not problems in the ACEC as much as they are restrictions of the trial systems discovered by running the ACEC. They could be considered portability problems if it were a goal to run the ACEC without modification on all validated compilation systems; however, since doing so would require the ACEC to avoid testing any system-dependent features and to code around errors in compilation systems, this has never been a goal of the ACEC.

3.3 POSSIBLE ENHANCEMENTS

There are several possible enhancements to the ACEC product which would increase the usefulness of the ACEC to users, making the ACEC easier and faster to use (particularly faster with respect to programmer time and effort) and to increase the coverage of ACEC.

- Merge ACEC with the AES.
 - Incorporate ideas on user interface from the AES.
 - > Interactive querying of status of testing.

What problems ran successfully, with errors, or were not attempted yet.

- > Interactive selection of sets of tests to be run.

- > Using results of early testing to automate later adaptations.
- Have Pretest query the user once as to what measurement technique to use.
- Incorporate selected tests or groups from AES.
 - > Adapt existing AES test problems and incorporate them into the ACEC.
 - > Consider classes of AES tests and add problems to cover the topic.
 - Test for implementation dependencies.
 - Collect data to determine run-time system size.
- Incorporate assessors for other environment capabilities.
 - > Profiler / Timing analysis tool
 - > Cross Reference / Interactive browser
 - > Test Coverage tool
 - > Test Bed Generator
 - > Pretty Printer
 - > Stub Generator
 - > Syntax Editor
 - > Assertion checker
 - > Name Expander
 - > Source Generator
- Enhance user documentation

There are several areas in which the user documentation might be enhanced. Some users have suggested that the ACEC documentation be expanded to include all issues which might affect compiler selection.

A hypertext version of the Guides and VDD would be valuable.
- Enhance the statistical robustness of the problem factor estimates computed by CA.

In the current CA program when comparing five (similar) systems, if the measurements from four are close and the fifth system is very much larger, the calculated mean will be far from all the systems and so ALL systems will be flagged as outliers. In such a case, it would be better if a more robust estimator of central location were used which would be closer to the

four similar systems and so only the fifth system would be flagged as an outlier. While using a median, as in the Release 2.0 analysis tool MEDIAN, would not permit the computation of confidence intervals, there are statistical techniques which can be used to compute an estimate of the mean which is less affected by one or two outliers. CA uses such a technique when calculating system factors.

- Enhance the statistical robustness behavior of CA with respect to missing data.

Where the set of systems being analyzed by CA contains a wide performance range of systems and for a few test problems one of the very slow systems has missing data, the computation of the mean for these problems will be greatly influenced by the missing data. The estimated means derived by ignoring the missing data can be very different from the estimates which would have been made if the data from the slow system were available. This different estimate for the problem mean can also affect the calculation of residuals and the identification of outliers.

The processing of the GENERIC group for the Summary Over All Groups Report for compile-time analysis shows this effect; data for the two slower compilation systems were missing and this resulted in ALL three other systems being flagged as very slow outliers. This example is also counter-intuitive in that all the systems are shown as slower than average, which suggests that the estimate for the average should be reduced.

- Extend coverage of ACEC assessors to test for reliability/robustness of a system by using random self-checking program generators (tailorable to emphasize different language areas). Observed failure rates for such programs can be used to estimate the number of errors in the compilation system.
- Provide more flexible reporting options for SSA, permitting users to select a range of report styles: from one similar to the AES with much textual descriptions and interpretation of results; to one similar to the current SSA report; to a very terse style listing only feature name and results.

The last form may permit a presentation format which lists results from several different systems on one report, facilitating comparisons between systems.

- Provide for easy comparisons of assessor results. One simple scheme for this would be to provide for a one line list of question name and results from different systems in separate columns on one page (this is applicable when results can be presented in a few columns).

It would be possible to perform some analysis on the assessor results data, even when it is not numeric (only reflecting YES/NO results) by highlighting questions where systems did not have the same response. If the answers to a question are all YES or all NO, the question does not differentiate between systems and may be omitted in a report designed to highlight differences between systems.

- Provide an assessor for compilation system reliability/robustness.

- Support for testing Ada 9X features and changing existing ACEC code which turns out to be in conflict with Ada 9X. This is applicable after 9X is adopted.

4. SUMMARY

Many of the changes made in Release 3.0 offer help in several areas:

- Ease of use is enhanced by providing a MENU for the analysis tools, the Pretest procedure, and the renaming to reflect test purposes which eases the selection of subsets.
- Capability is extended by providing a Capacity Assessor and additional performance tests.
- The utility of the statistical analysis is enhanced by providing confidence intervals for system factors.

Running the ACEC Release 3.0 on the trial systems demonstrated that not all validated Ada compilations systems will accept all valid Ada programs, even excluding those with "obvious" system dependencies such as tying tasks to interrupts. Ada compilers are large and complex programs and it should not be too surprising that they are not error-free. While the ACEC was not designed to test correctness of compilers, it has (since Release 1.0) been effective in discovering errors in implementations. This should be considered as a beneficial side-effect of the ACEC to the extent that it has encouraged compiler implementors to correct errors.

Having Independent Validation and Verification (IV&V) has been helpful to the ACEC program. Comments from experienced beta testers have permitted enhancements to be made to the product before general release by exposing the product to individuals with a different professional perspective and background than the developers. Especially with respect to documentation, an independent reviewer can call attention to issues which seem too obvious to the developers to require explanation or where descriptions are not clear to individuals not already familiar with the product. ACEC testers have used different compilation systems and have pointed out issues which were not problems on the trial systems Boeing used.

The input from the external reviewers has resulted in changes which have improved the ACEC product by enhancing usability, clarity of documentation, accuracy of tests, scope of coverage of the suite, and the utility of analysis program results.

While there remain areas of possible enhancement, as discussed in the prior section, Release 3.0 provides a solid baseline for Ada compilation system evaluation which is both easier to use than the prior releases and provides enhanced functional capabilities.

5. NOTES

This section contains information only and is not contractually binding.

5.1 ABBREVIATIONS AND ACRONYMS

ACEC	Ada Compiler Evaluation Capability
Ada 9X	Future MIL_STD_1815B
CA	Comparative Analysis
CPU	Central Processing Unit
I/O	Input / Output
LRM	(Ada) Language Reference Manual (MIL-STD-1815A)
SSA	Single System Analysis
TOR	Technical Operating Report
VDD	Version Description Document